

## ГЛАВА XX.

### Сложность алгоритмов, переобучение, смещение и разброс

*Нельзя передать сложность,  
а только знание о ней.*

А. Перлис

*Чем человек сложнее,  
тем интереснее... и опаснее.  
из сериала «Vikings»*

Обсудим несколько фундаментальных понятий машинного обучения. Первое – **переобучение**, также говорят **переподгонка** или **перенастройка (overfitting)**, – явление, когда ошибка на тестовой выборке заметно больше ошибки на обучающей. Это главная проблема машинного обучения: если бы такого эффекта не было (ошибка на тесте примерно совпадала с ошибкой на обучении), то всё обучение сводилось бы к минимизации ошибки на обучении (т.н. эмпирическому риску). На рис. XX.1 (слева) выборка на рисунке классифицирована без ошибок (можно догадаться, что решающим деревом), но также понятно, что ошибка на достаточно большом тесте (выборке, которая будет распределена также как обучающая) не будет нулевой (слишком близко к объектам одного из классов располагается разделяющая поверхность, да и очевидное линейное разделение оно не осуществляет).

Переподгонка более удачный термин, т.к. fit – это как раз подгоняться или настраиваться.

Второе фундаментальное понятие – **недообучение**, также говорят **недоподгонка** или **недонастройка (underfitting)**, – явление, когда ошибка на обучающей и тестовой выборках достаточно большая, часто говорят «**не удаётся настроиться на выборку**<sup>1</sup>». На рис. XX.1 (справа) ошибка классификации соответствует ошибке случайного решения: примерно 50% неверных ответов, очевидно, что выбранная для классификации модель

<sup>1</sup> Такие странные термины можно объяснить обучением итерационными методами, например градиентного бустинга (на каждой итерации добавляем новый базовый алгоритм) или нейронной сети (на каждой итерации делаем SGD по одному батчу данных). Тогда недообучение можно наблюдать, когда делаем слишком мало итераций: «не успели обучиться», а переобучение – когда слишком много.

алгоритмов (линейная) не соответствует данным и на тестовой выборке также ошибка будет около 50%.

Третье понятие – **сложность (complexity) модели алгоритмов**. Сложность допускает множество формализаций, но, как правило, оценивает, насколько разнообразно семейство алгоритмов в модели с точки зрения их функциональных свойств (например, способности настраиваться на выборки). Как мы убедимся, повышение сложности (т.е. использование более сложных моделей) решает проблему недообучения и вызывает переобучение. Например, на рис. XX.1 справа модель слишком простая (линейная), поэтому её не удаётся «настроить на данные», также часто говорят «**бедная модель**».

Сложное = достаточно  
разнообразное.

Отметим, что все описанные понятия связаны с тем, как соотносятся ошибки на обучении и тесте. Эти ошибки также могут различаться из-за того, что обучающая и тестовая выборки по-разному распределены из-за:

- неправильного разбиения данных на обучение и контроль,
- изменением данных со временем (и необходимостью предсказывать будущее),
- особенности теста, например, данные для теста собраны с помощью другой аппаратуры, при других условиях и т.п.

Будем предполагать, что всех этих проблем нет: обучение и тест распределены одинаково.

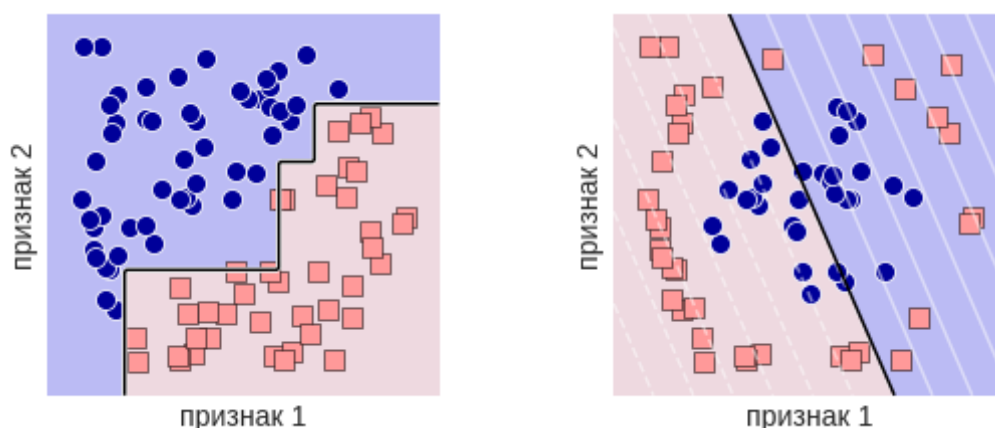


Рис. XX.1. Пример переобучения (слева) и недообучения (справа).

Сначала опишем на примере, как проявляется проблема выбора сложности и почему возникает переобучение. Для начала рассмотрим задачу регрессии. Для простоты будем считать, что это регрессия от одного признака. Целевая

зависимость  $y(x)$  известна в конечном наборе точек. На рис. XX.2 показана выборка для зависимости вида  $y = \sin(x) + \varepsilon$ , где  $\varepsilon$  – шум, на рис. XX.3 – для зашумлённой пороговой зависимости  $y = I[x \geq 0.5] + \varepsilon$ .

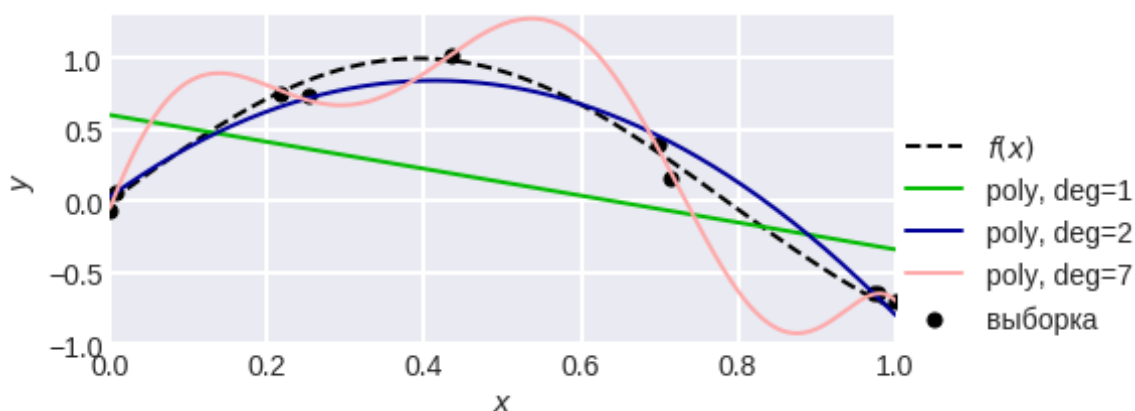


Рис. XX.2. Настройка полиномов различных степеней на  $\sin(x)$ .

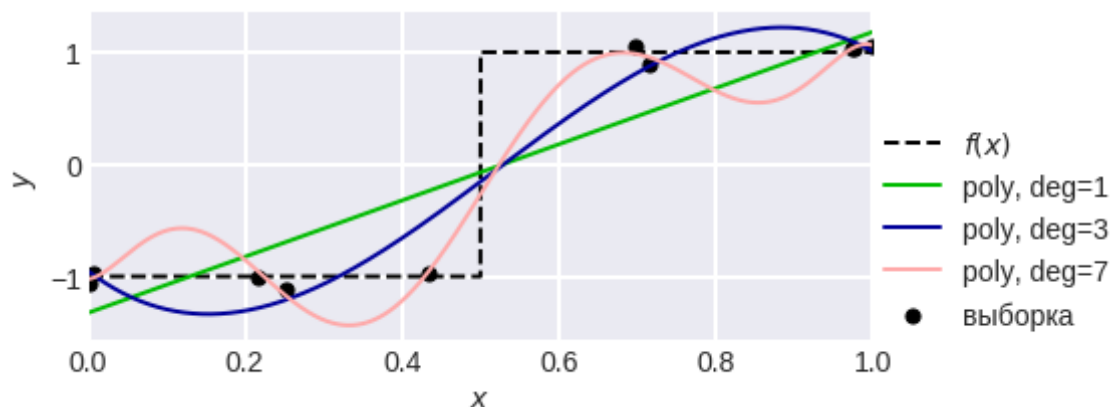


Рис. XX.3. Настройка полиномов различных степеней на  $I[x \geq 0.5]$ .

На рисунках показаны также решения указанных задач полиномиальной регрессией с разными степенями полиномов. Видно, что в обеих задачах полином первой степени явно плохо подходит для описания целевой зависимости, второй или третьей — достаточно хорошо её описывает, хотя ошибки есть и на обучающей выборке, седьмой — идеально проходит через точки обучающей выборки, но совсем не похож на «естественную функцию» и существенно отклоняется от целевой зависимости в остальных точках.

Если попробовать решить задачу полиномами различной степени, то мы получим рис. XX.4 (он построен для первой задачи, но во второй картина аналогичная). Видно, что с увеличением степени ошибка на обучающей выборке падает, а на тестовой (в качестве тестовых объектов взяли точки мелкой равномерной сетки отрезка  $[0, 1]$ ) — сначала падает, потом возрастает. Также обратим внимание, что ошибки на обучающей и тестовой выборках

были почти одинаковы для полиномов степени  $k \leq 3$ , а далее при увеличении степени растёт разница между ошибками: усиливается переобучение.

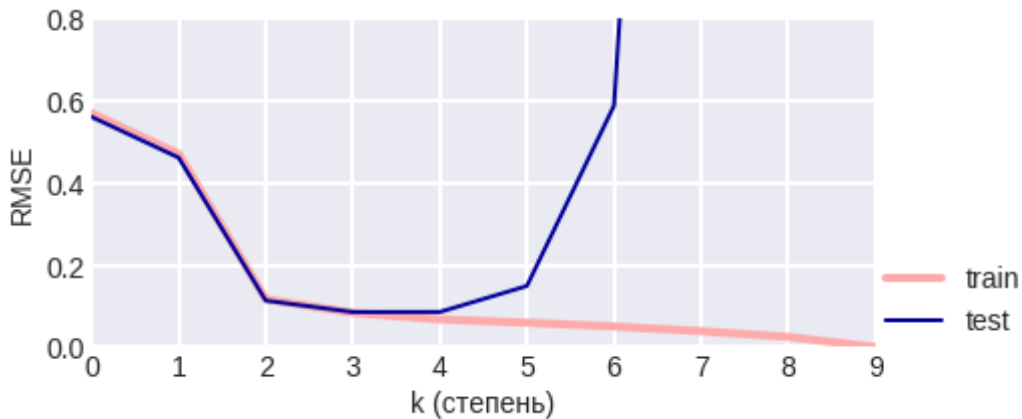


Рис. XX.4. Зависимость ошибки на обучении и тесте от степени полинома.

Попробуем разобраться, в чём дело с теоретической точки зрения. Наша целевая зависимость имеет вид

$$y(x) = f(x) + \varepsilon(x), \quad \varepsilon(x) \sim \text{random}(0, \sigma^2),$$

есть истинная зависимость  $y = f(x)$ , но в точках обучающей выборки она известно с точностью до шума  $\varepsilon(x)$ , в каждой точке шум свой, шумы в разных точках независимы и одинаково распределены со средним 0 и дисперсией  $\sigma^2$ .

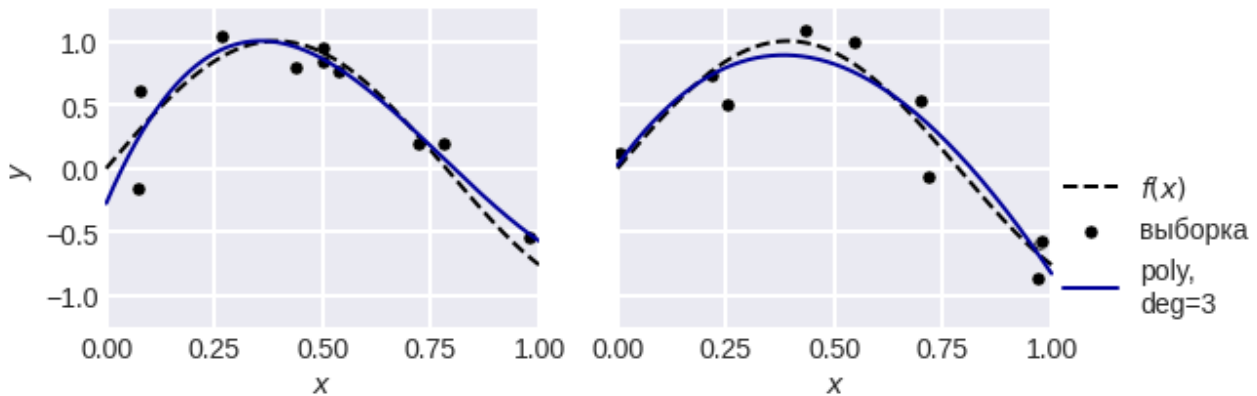


Рис. XX.5. Две случайные обучающие выборки, описывающие одну зависимость, и регрессионные решения, которые им соответствуют.

На рис. XX.5 показаны две случайные обучающие выборки для одной и той же целевой зависимости и два разных полинома, которые получились в результате решения указанных задач полиномиальной регрессии (каждый обучался по своей выборке). Не смотря на схожий вид построенных полиномов, в некоторых точках их значения сильно различаются (см., например,  $x = 0$ ).

Дальше мы будем брать математическое ожидание квадрата отклонения ответа регрессора  $a(x)$  от истинного значения в фиксированной точке  $x$

$$\mathbf{E}_{(x_i, f(x_i) + \varepsilon_i)_{i=1}^m} (y(x) - a(x))^2$$

по всем данным (обучающим выборкам, отражающим зависимость  $y = f(x)$ ) и всем настройкам алгоритма (некоторые алгоритмы имеют стохастическую природу, например случайный лес).

Для упрощения формул не будем указывать конкретную точку:  $y \equiv y(x)$ ,  $a \equiv a(x)$ . Итак, обозначенное выше матожидание равно

$$\begin{aligned} \mathbf{E}(y - a)^2 &= \mathbf{E}(y^2 + a^2 - 2ya) = \\ &= \mathbf{E}y^2 - (\mathbf{E}y)^2 + (\mathbf{E}y)^2 + \mathbf{E}a^2 - (\mathbf{E}a)^2 + (\mathbf{E}a)^2 - 2\mathbf{E}ya = \\ &= (\mathbf{E}y^2 - (\mathbf{E}y)^2) + (\mathbf{E}a^2 - (\mathbf{E}a)^2) + ((\mathbf{E}y)^2 - 2\mathbf{E}ya + (\mathbf{E}a)^2) = \\ &= \mathbf{D}y + \mathbf{D}a + (\mathbf{E}y)^2 - 2\mathbf{E}ya + (\mathbf{E}a)^2 = \\ &= \mathbf{D}y + \mathbf{D}a + f^2 - 2f\mathbf{E}a + (\mathbf{E}a)^2 = \\ &= \mathbf{D}y + \mathbf{D}a + (f - \mathbf{E}a)^2. \end{aligned}$$

Синим выделено добавление и вычитание одинаковых слагаемых. Красным – такой переход

$$\mathbf{E}ya = \mathbf{E}((f(x) + \varepsilon(x)) \cdot a(x)) = \mathbf{E}(f(x) \cdot a(x)) + \mathbf{E}(\varepsilon(x) \cdot a(x)),$$

$f(x)$  по сути является константой (значение истинной зависимости в точке  $x$ ), а последнее слагаемое равно произведению матожиданий, поскольку шум в точке и ответ алгоритма в точке независимы (предполагается, что точка берётся из теста, а алгоритм конечно же зависит от шума, но на объектах из обучения):

$$\mathbf{E}(\varepsilon(x) \cdot a(x)) = \mathbf{E}(\varepsilon(x)) \cdot \mathbf{E}(a(x)) = 0 \cdot \mathbf{E}(a(x)) = 0.$$

**Разбросом (variance)** назовём дисперсию ответов алгоритмов  $\mathbf{D}a$ , а **смещением (bias)** – матожидание разности между истинным ответом и выданным алгоритмом:  $\mathbf{E}(y - a) = f - \mathbf{E}a$ . Мы получили, что ошибка (точнее её матожидание) раскладывается на три составляющие:

$$\mathbf{E}(y - a)^2 = \sigma^2 + \text{variance}(a) + \text{bias}^2(f, a), \quad (\text{XX.1})$$

шум, разброс и смещение (точнее квадрат смещения). Все они вычислены в конкретной точке, но при желании, можно проинтегрировать формулы по всем объектам (точнее, по какому-то распределению всех объектов) и получить уже смещение и разброс модели алгоритмов как таковой.

$$\text{Ошибка} = \text{шум (шум в данных)} + \text{разброс (разнообразие ответов модели)} + \text{смещение}^2 \text{ (отклонение от истины)}$$

Слагаемое «шум»  $\sigma^2$  связано с шумом в самих данных, а вот два остальных – связаны с используемой моделью алгоритмов. Понятно, что **разброс характеризует разнообразие алгоритмов** (из-за случайности обучающей выборки, в том числе шума, и стохастической природы настройки), а **смещение – способность модели алгоритмов настраиваться на целевую зависимость**. Проиллюстрируем это. На рис. XX.6-7 – показаны различные полиномы первой степени, они настроены на разных обучающих выборках. В точке  $x=0.5$  ответы алгоритмов являются случайными величинами, они немного «разбросаны» (есть variance), а также они сильно смещены (есть bias) относительно правильного ответа (который, кстати, даже если нам и известен, то с точностью до шума). На рис. XX.7. показан «срез» предыдущего рисунка по значению  $x=0.5$ : синие точки – значения построенных полиномов в точке  $x$ , чёрные целевое значение с точностью до шума, стрелками показаны значения шума, разброса и смещения.

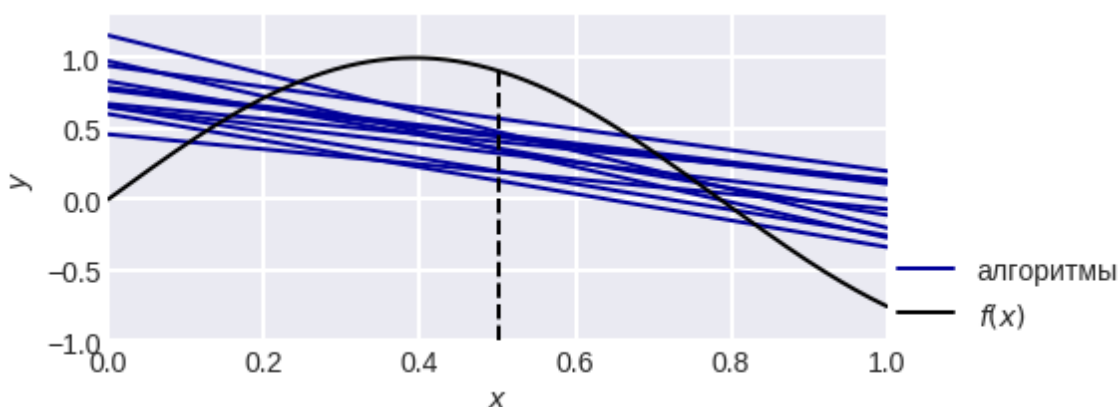


Рис. XX.6. Полиномы 1й степени, настроенные на разных обучающих выборках.

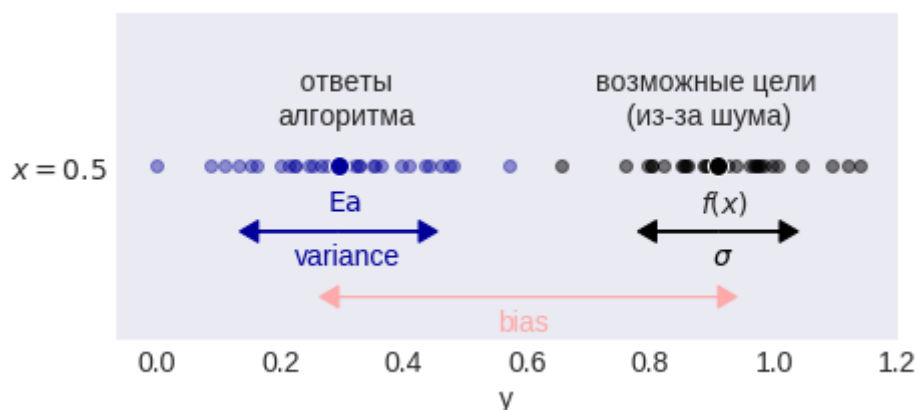


Рис. XX.7. Шум, разброс и смещение при настройках полиномов 1й степени.

На рис. XX.8-9 изображены уже полиномы второй степени (настроенные на тех же выборках). В точке  $x=0.5$  у них сильно меньше смещение и чуть меньше разброс. Видно, что они совсем неплохо описывают целевую зависимость во всех точках.

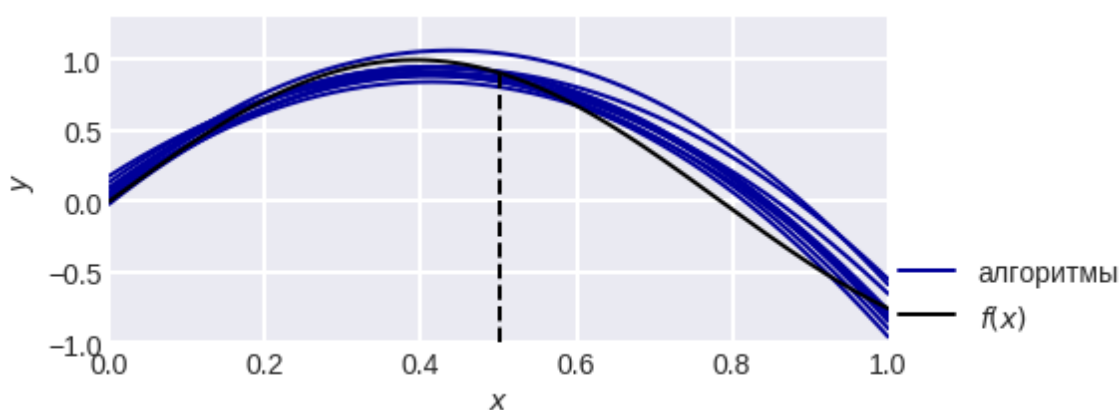


Рис. XX.8. Полиномы 2й степени, настроенные на разных обучающих выборках.

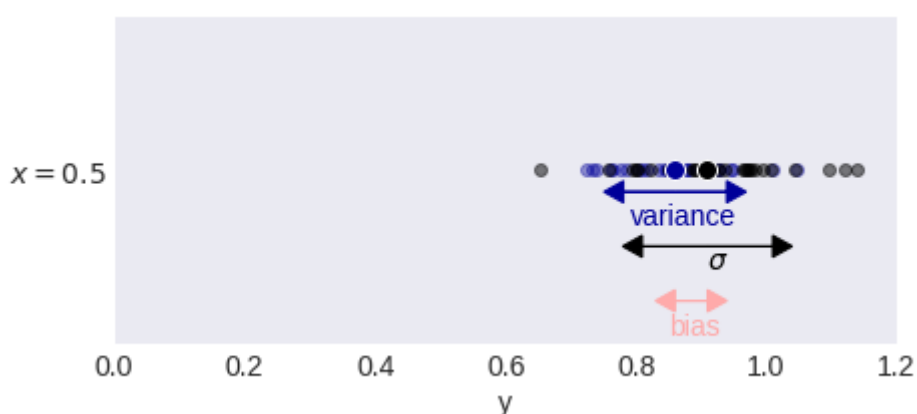


Рис. XX.9. Шум, разброс и смещение при настройках полиномов 2й степени.

Часто разброс и смещение иллюстрируют такими картинками – см. рис. XX.10. Если провести аналогию, что модель алгоритмов – это игрок в дартс, то самый

лучший игрок будет иметь небольшое смещение и разброс – его дротики ложатся кучно «в яблочко», если у игрока большое смещение, то они сгруппированы около другой точки (вдали от центра мишени), а если большой разброс, то они ложатся совсем не кучно. Эта аналогия понятна, но может сбить с толку. Например, известно, что спортсменов учат стрелять кучно, поскольку нужного смещения легко добиться (целясь с поправкой, например, если попадаете ниже яблочка, то надо целиться выше). С алгоритмами машинного обучения всё сложнее. Смещение нельзя просто «подвинуть»: на рис. XX.6 видно, что смещение в каких-то точках положительное, а в каких-то отрицательное и прибавка поправки не улучшит модель. Поэтому для устранения смещения приходится переходить к другой (например, более сложной) модели, а у неё может быть уже другой разброс.





	Малое смещение	Большое смещение
Малый разброс		
Большой разброс		

Рис. XX.10. Объяснение разброса и смещения на примере игры в дартс.

Рассмотрим рис. XX.11, который часто приводят при объяснении разброса и смещения, он полностью согласуется с рис. XX.4. При увеличении сложности модели (например, степени полинома) ошибка на независимом контроле сначала падает, потом начинает увеличиваться. Обычно это связывают с уменьшением смещения (в сложных моделях очень много алгоритмов, поэтому наверняка найдутся те, которые хорошо описывают целевую зависимость) и увеличением разброса (в сложных моделях больше алгоритмов, а следовательно, и больше разброс).



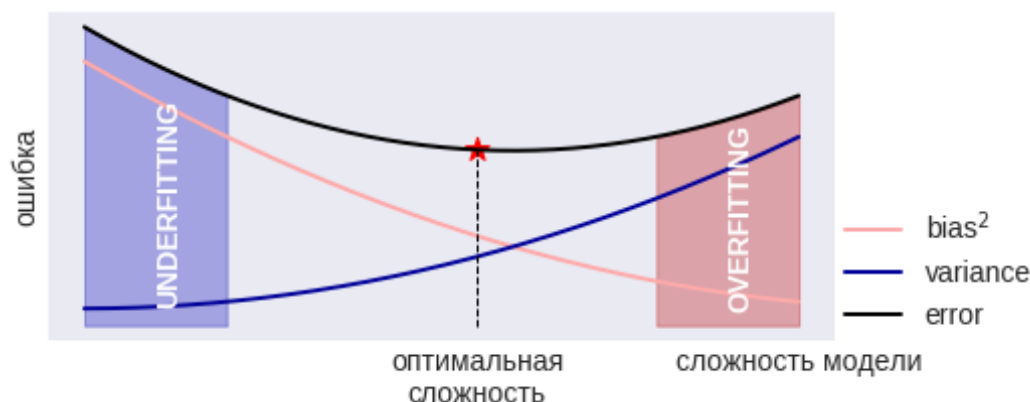


Рис. XX.11. Классическая иллюстрация изменения разброса и смещения.

Для простых моделей характерно недообучение вызванное большим смещением: они не могут хорошо описать целевую зависимость, поэтому смещение большое. Для сложных моделей характерно переобучение вызванное большим разбросом: алгоритмов в модели слишком много, при настройке мы выбираем ту, которая хорошо описывает обучающую выборку, но из-за сильного разброса она может допускать большую ошибку на тесте.

Кстати, иногда сложность модели определяют через разброс:

$$\text{complexity} = \text{variance}, \quad (\text{XX.2})$$

что кажется вполне интуитивным. Правда, разброс считается в конкретной задаче, но и сложность по логике должна зависеть от задачи. Также отметим, что разброс тогда надо агрегировать по всем объектам.

Теперь рассмотрим задачу классификации. Отметим, что для неё тоже есть результат о разложении ошибки на шум, разброс и смещение<sup>1</sup>. На рис. XX.12-14 показаны результаты экспериментов в задаче с двумя классами (стандартный набор данных «два полумесяца») и моделью  $k$  ближайших соседей (kNN) при разных  $k$ . Результат также согласуется с рис. XX.11, если учесть, что изображена точность, а не ошибка (т.е. чем больше, тем лучше алгоритм), и сложность алгоритма  $\sim 1/k$  (чем больше  $k$ , тем проще алгоритм). Возникает вопрос, а почему так вводится сложность для kNN? Ведь при разных  $k$  эти алгоритмы

- имеют одинаковые параметры (а точнее, не имеют параметров, только гиперпараметр  $k$ ; рассматриваем алгоритм в евклидовой метрикой),

<sup>1</sup> P. Domingos A unified bias-variance decomposition and its applications //17th International Conference on Machine Learning. – 2000. – С. 231-238.

- требуют хранения всей обучающей выборки (являются «ленивыми» – lazy algorithms),
- 9NN даже «чуть сложнее в реализации», чем 1NN (требует поиска большего числа соседей).

Здесь можно сослаться на (XX.2). На рис. XX.13 показаны разделяющие поверхности метода 1NN для разных выборок, которые описывают одну и ту же целевую зависимость. Они очень сильно отличаются друг от друга. Разделяющие поверхности kNN при больших  $k$ , см. рис. XX.14 почти не различаются. И чем выше  $k$ , тем стабильней результат<sup>1</sup>.

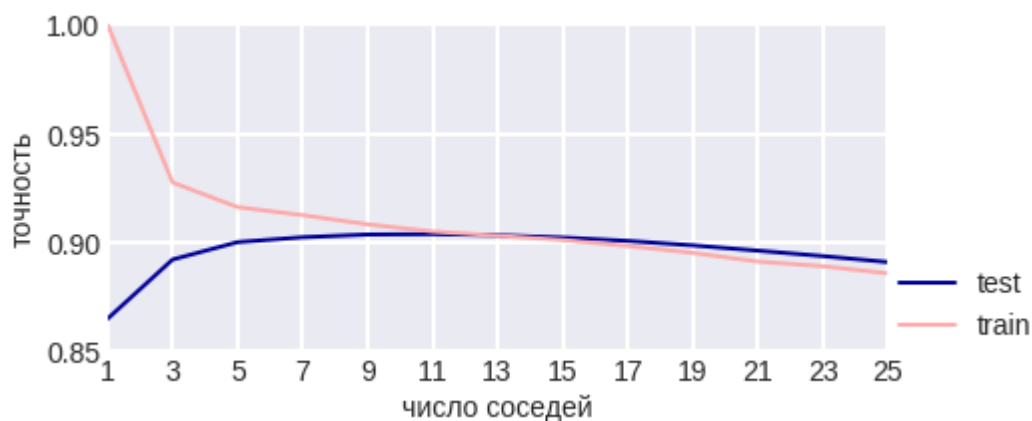


Рис. XX.12. Точность метода kNN при разных  $k$  на обучении и контроле.

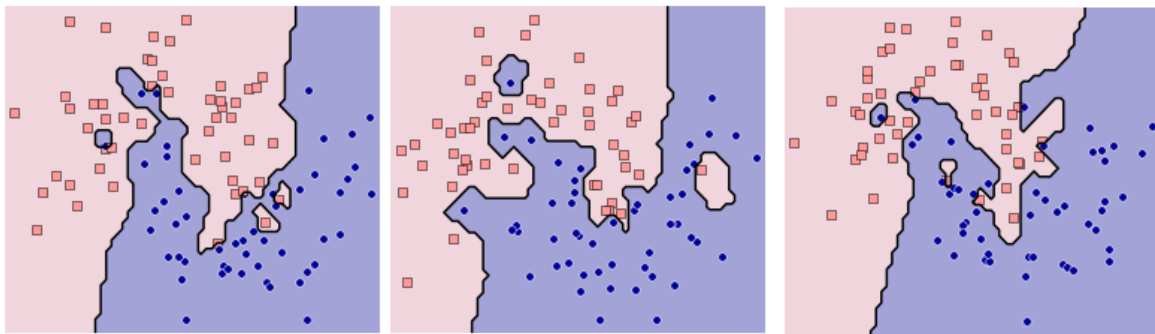


Рис. XX.13. Разделяющие поверхности 1NN для разных выборок (одинаково распределённых).

<sup>1</sup> При больших  $k$  алгоритм становится константным: всё относит к большему классу, т.е. становится совсем простым: перестаёт зависеть от выборки.

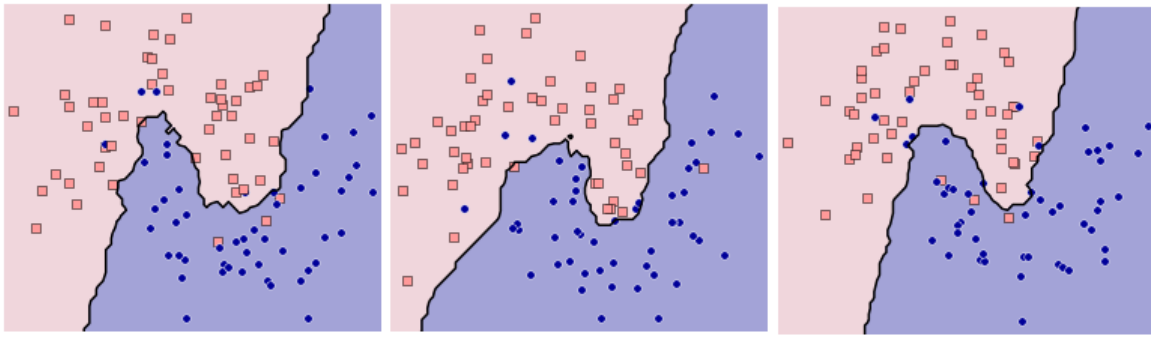


Рис. XX.14. Разделяющие поверхности 9NN для тех же выборок.

Теперь покажем, что на практике рис. X.11 наблюдается не в точности, о чём часто умалчивают стандартные учебники по машинному обучению. Проведём эксперименты по оцениванию разброса и смещения в модельных задачах. Для этого надо определить целевую зависимость, способ формирования обучающих выборок. Сгенерировать несколько обучающих выборок

$$D_1 = (x_i^1, y_i^1)_{i=1}^m, \dots, D_k = (x_i^k, y_i^k)_{i=1}^m,$$

на каждой обучить алгоритм  $a_1(x), \dots, a_k(x)$ , оценить средний алгоритм:

$$a(x) = \frac{1}{k}(a_1(x) + \dots + a_k(x)),$$

разброс и смещение:

$$\text{variance}(a(x)) \approx \mathbf{D}\{a_i(x)\}_{i=1}^k,$$

$$\text{bias}(f(x), a(x)) \approx f(x) - \mathbf{E}a(x).$$

На рис. XX.15-16 приведены результаты для задачи с пороговой целевой зависимостью, а на рис. XX.17-18 для задачи с целевой зависимостью  $y = \sin(4x)$ .

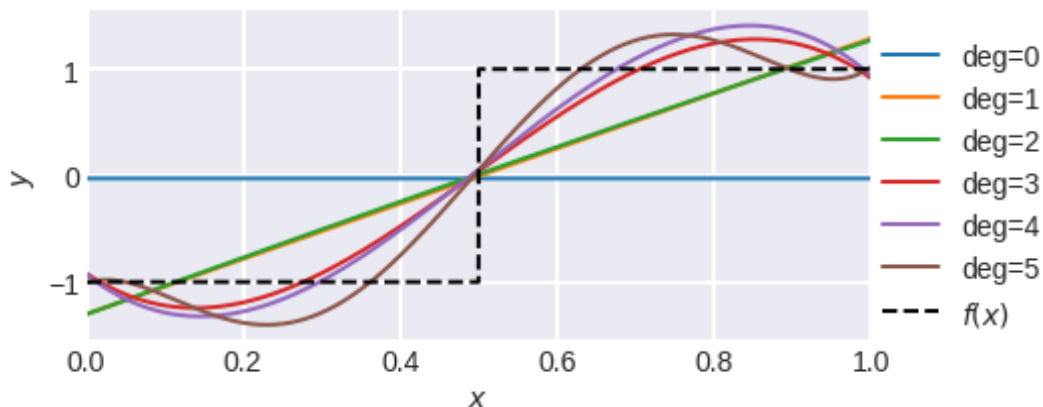


Рис. XX.15. «Средние» полиномы различных степеней в задаче «ступенька».



Рис. XX.16. Ошибка при разных степенях полиномов в задаче «ступенька».

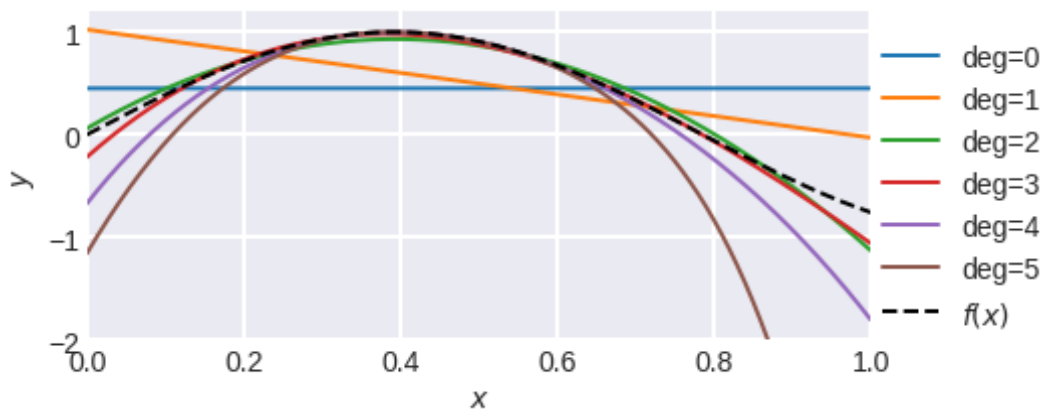


Рис. XX.17. «Средние» полиномы различных степеней в задаче «синус».

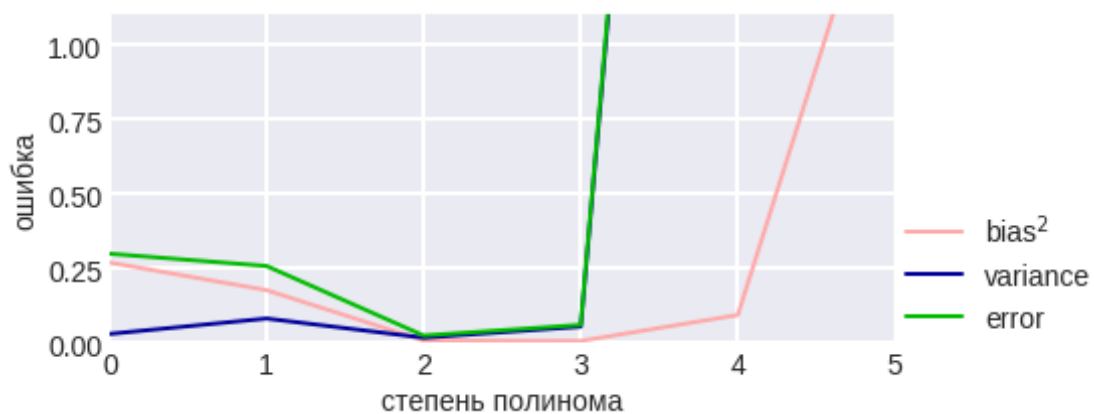


Рис. XX.18. Ошибка при разных степенях полиномов в задаче «синус».

Очевидно, что степень полинома – очень естественная мера сложности для полиномиальной регрессии. Но полученные рисунки немного отличаются от рис. XX.11:

- общая ошибка может быть не совсем унимодальна от сложности,
- смещение и разброс могут не быть строго монотонными,

- смещение может возрастать при увеличении сложности (подумайте, так ли это).

Почему так происходит? Одна из причин в том, что «сложность модели» правильнее определять для конкретных данных! Например, ступенчатая функция нечётная (с точностью до смещения) и для восстановления такой целевой зависимости лучше подходят полиномы с нечётной старшей степенью. Кстати, «средние полиномы», а это как раз  $Ea(x)$ , степени 1 и 2 на рис. XX.15 совпадают.

Модель сложная или простая в конкретной ситуации – для конкретных данных!

### Способы борьбы с переобучением

Кратко перечислим основные способы уменьшения эффекта переобучения. В качестве иллюстрации везде рассматривается задача регрессии, изображённая на рис. XX.2.

**1. Использование выборки специальной структуры.** Если выборка нам не задана априорно, а мы можем выбрать  $x_i$ , для которых «нам сообщат»  $y_i = y(x_i)$  (некоторый вариант т.н. активного обучения – active learning), то за счёт выбора  $x_i$  можно уменьшить переобучение. В рассматриваемой модельной задаче выбор точек в виде равномерной сетки существенно улучшает возможности использовать полиномы больших степеней для решения задачи регрессии, см. рис. XX.19. На практике, если есть возможность сформировать обучающую выборку (решить, каким клиентам предлагать услугу, какое подмножество документов оценивать ассессорам, где бурить скважину для взятия проб и т.п.), то этим надо пользоваться!

Если можно выбрать обучающую выборку – выбирайте с умом!

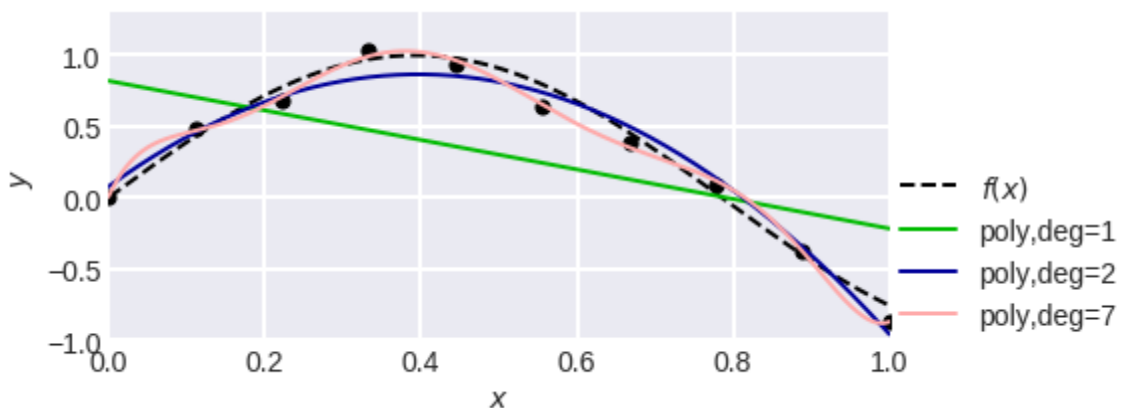


Рис. XX.19. Равномерная сетка точек в качестве обучающей выборки.

**2. Увеличение объёма данных.** Чем больше данных, тем более сложные модели можно на них

Ещё раз: данные первичны,  
модели вторичны!

строить – это давно известный принцип, но отметим, что для использования некоторых моделей может потребоваться слишком много данных. На рис. 20 при увеличении обучающей выборки использование полиномов 7й степени не приводит к переобучению.

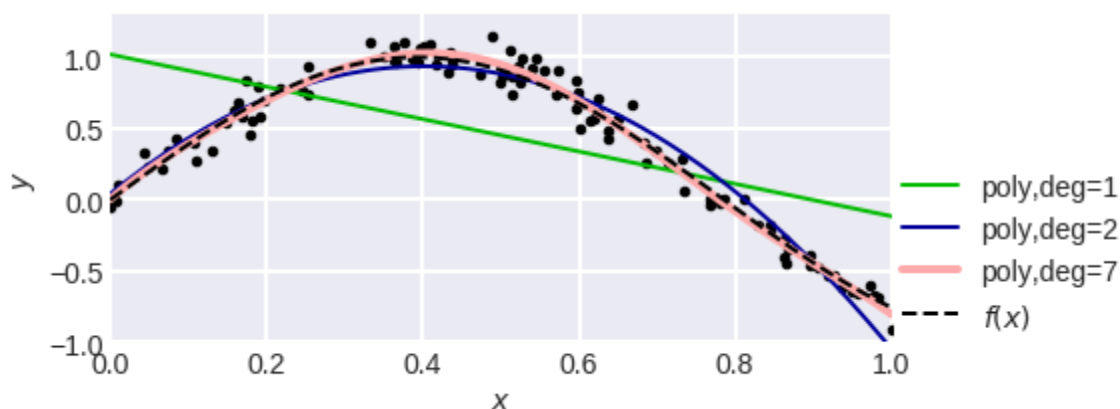


Рис. XX.20. Увеличенная обучающая выборка.

**3. Аугментация.** По сути, это использование идей предыдущих пунктов. Когда надо нарастить объём данных, но новых данных нет, их создают искусственно. На рис. XX.21 мы просто продублировали точки обучающей выборки, немного размазав их по оси  $x$ . Логика простая: данные нам даны с шумом, мы точно не знаем пары  $(x, y)$ , поэтому если немного сместим по  $x$ :  $(x \pm \varepsilon, y)$ , то хуже не будет (мы даже, может быть, компенсируем шум). Видно, что это позволило успешней использовать полином большой степени.

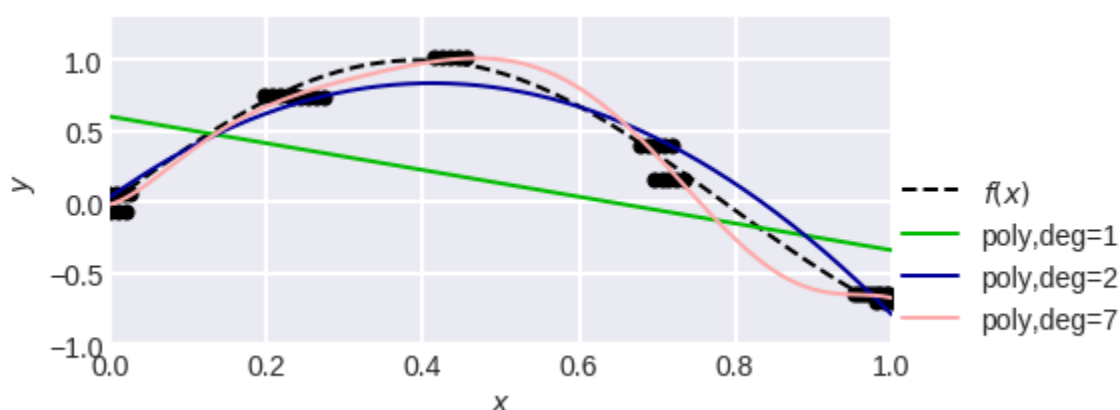


Рис. XX.21. Аугментированная обучающая выборка: добавляем несколько смещённых копий каждого объекта.

Вообще, аугментация – это увеличение выборки с помощью преобразований её элементов. На практике этот приём используют и для повышения качества решения задачи в целом. При классификации изображений, если на картинке

изображён некоторый объект, то он там останется при небольших сдвигах изображения, поворотах, размытии и зашумлении. Все эти преобразования можно использовать, чтобы пополнить обучающую выборку новыми объектами, что также позволит создать алгоритм, устойчивый к подобного рода преобразованиям.

Робастность алгоритма  
можно обеспечить  
преобразованиями данных.

**4. «Улучшение качества данных».** Под улучшением качества понимается уменьшение/устранение: шума, выбросов, дубликатов и пропусков. Строго говоря, это не способ борьбы с переобучением, а уменьшение шума в формуле для разложения ошибки алгоритма (XX.1). На рис. XX.22 показана такая же выборка, что и на рис. XX.20, но с большим шумом, соответственно здесь у полинома 7й степени и ошибка больше, даже при большом объёме данных.

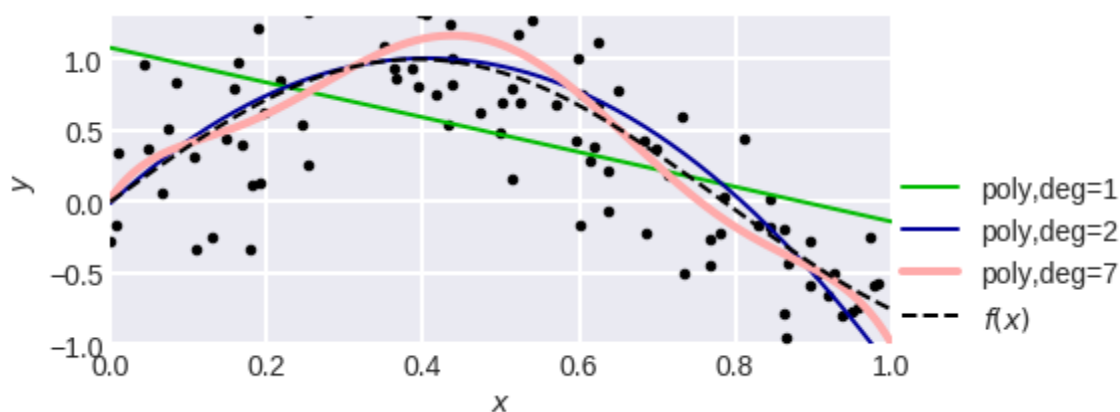


Рис. XX.22. Увеличение шума в обучающей выборке.

**5. Уменьшение избыточности данных.** Ещё один способ, который стоит упомянуть, заключается в устранении избыточности данных. Как правило, используют методы сокращения размерности признакового пространства или отбор (селекцию) признаков. На рис. XX.23 представлен результат следующего эксперимента: в модельной задаче число объектов и признаков  $m = n = 100$ , целевой признак выражается через первые два с небольшим шумом:

$$y = X_1 - X_2 + \text{norm}(0, 0.5),$$

все признаки случайные  $X_i = \text{norm}(0, 1)$ ,  $i \in \{1, 2, \dots, n\}$ . На графике показана ошибка на обучении и тесте при различном числе признаков  $t$ , при этом используются первые  $t$  признаков. Естественным образом, наименьшая ошибка на тесте при  $t = 2$ , при увеличении числа признаков эта ошибка также возрастает и наблюдается переобучение, но ошибка на обучении уменьшается и достигает



нуля при  $t = m = n^1$ . Этот график нам напоминает аналогичный, где показана зависимость от сложности модели (в линейной модели чем больше признаков, тем больше коэффициентов и тем больше сложность).



Рис. XX.23. Ошибка на обучении и тесте при разном числе признаков.

**6. Использование других данных / задач / готовых моделей.** Этот приём сложно проиллюстрировать на примере оставаясь в рамках классического машинного обучения. Чаще всего он используется в глубоком обучении: сначала нейросеть обучают на аналогичной задаче, в которой есть много размеченных данных (их можно дёшево достать или синтезировать), затем на исходной задаче её «дообучают» (это так называемое **трансферное обучение – transfer learning**). Дообучить – значит немного поменять параметры, чтобы функциональность сети изменилась не сильно, но качество на обучающей выборке выросло.

Интересное наблюдение: первые 5 способов борьбы с переобучением, меняют данные, а не модель.

**7. Регуляризация.** В широком смысле, регуляризация – это **изменение процесса обучения (настройки алгоритма), которая позволяет уменьшить эффект переобучения<sup>2</sup>**. Как правило, заключается в создании некоторых искусственных ограничений при настройке модели и, как следствие, уменьшении её сложности:

- использование дополнительных штрафующих слагаемых в задаче оптимизации (например, L2-слагаемое в Ridge-регрессии),
- переход к более простой модели (подрезка решающих деревьев),

<sup>1</sup> Нетрудно догадаться, что при случайной квадратной признаковой матрице п.в. ошибка будет нулевая.

<sup>2</sup> В последние годы всё, что приводит к уменьшению переобучения, стали называть регуляризацией, что кажется не очень правильным (например, увеличение выборки с помощью сбора новых данных), но такова тенденция в терминологии.



- ограничения при модификации параметров в процессе обучения (dropout в нейронных сетях),
- уменьшение числа параметров (факторизация матрицы параметров, разделение параметров).

На рис. XX.24 показано использование Ridge-регрессии при настройке полиномов (полиномиальной регрессии), т.е. использование штрафующего слагаемого, равного L2-норме весов модели.

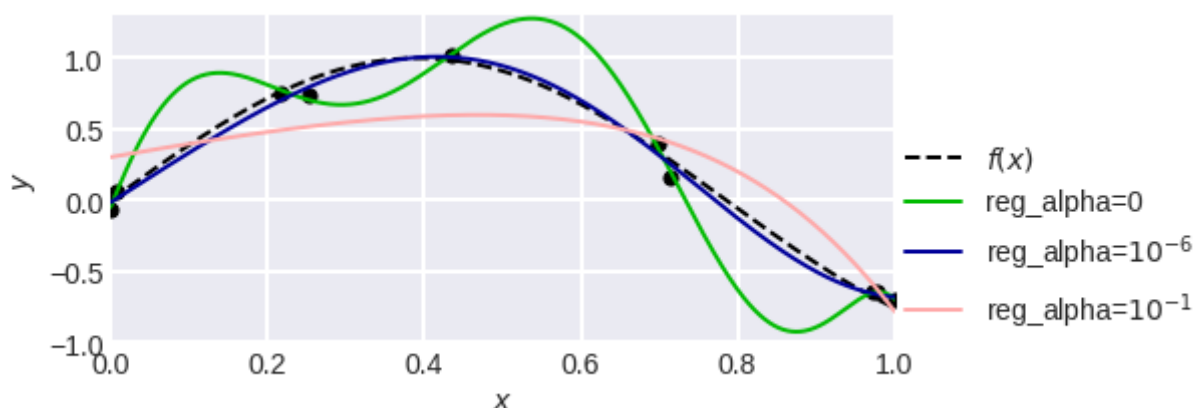


Рис. XX.24. Полином большой степени при разных коэффициентах L2-регуляризации.

На рис. XX.25 показаны изменения ошибки, квадрата смещения и разброса при изменении параметра регуляризации (уже на другой задаче: «ступеньке», по горизонтали параметр уменьшается). Уменьшение параметра соответствует увеличению сложности модели – разброс увеличивается. Понятно, что процедура валидация, которая нужна для настройки гиперпараметров, в том числе нужна для определения оптимального коэффициента регуляризации, при котором ошибка минимальна.

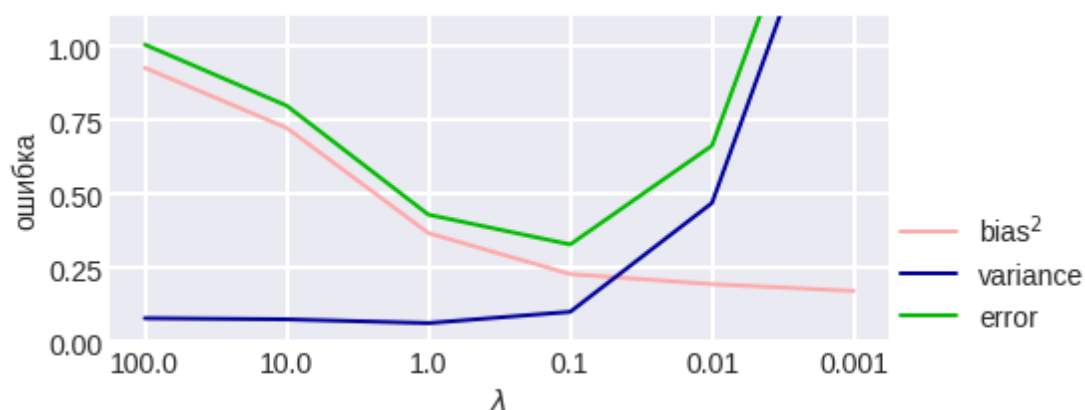


Рис. XX.25. Зависимость ошибки от коэффициента регуляризации.

На рис. XX.26 показаны решения без регуляризации (с нулевым коэффициентом регуляризации) и с регуляризацией (с небольшим ненулевым коэффициентом) на одной из модельных задач, которую мы рассматривали в этой главе. Обозначен коридор стандартного отклонения ответов моделей – его ширина говорит о разбросе.

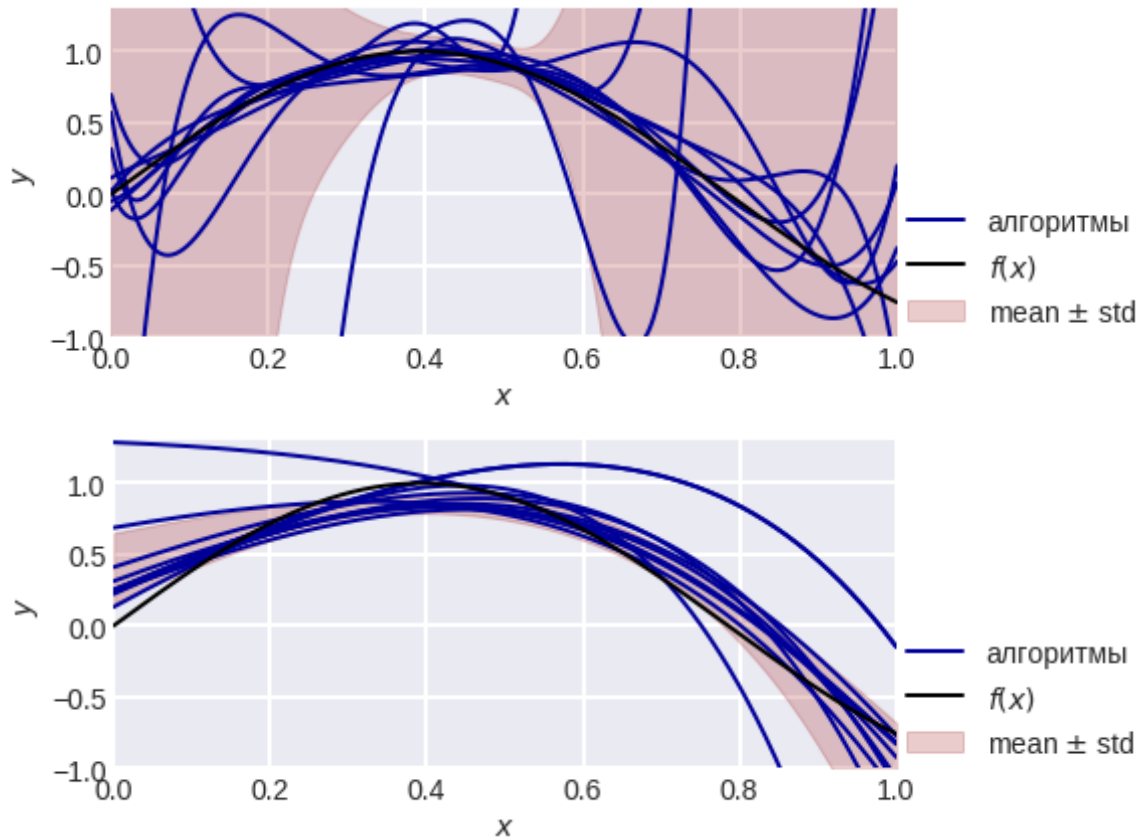


Рис. XX.26. Получаемые полиномиальные регрессии без регуляризации (вверху) и с регуляризацией (внизу).

**8. Организация валидации.** Про это мы подробно говорили в главе «**Выбор модели**». Чтобы не переобучиться, надо следить за ошибкой на обучении и тесте (и предпринимать описанные выше действия, если эти ошибки сильно отличаются). Обратим внимание на приём **ранней остановки (early stopping)** обучения. Если алгоритм обучается итерационно, то часто следует выбрать итерацию, на которой следует прекратить обучение. На рис. XX.27 показана ошибка на отложенном контроле при обучении бустинга – после отметки  $n\_estimators = 30$  при увеличении числа базовых алгоритмов в ансамбле ошибка увеличивается. Аналогичный эффект наблюдается в нейронных сетях при увеличении числа итераций в градиентном методе обучения.

В бэггинге и случайном лесе такого эффекта нет: чем больше алгоритмов в ансамбле, тем не хуже.

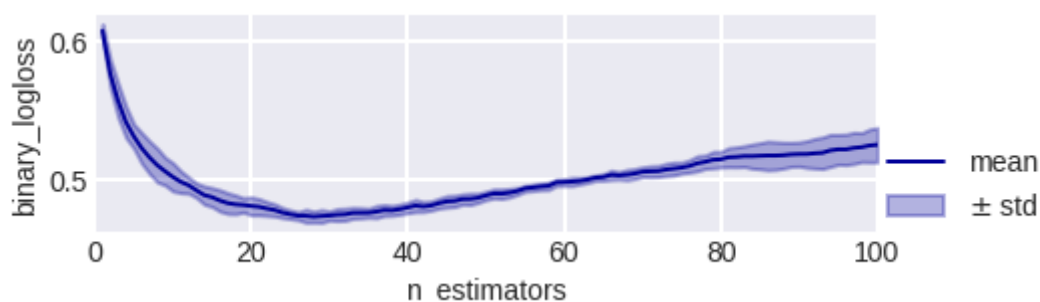


Рис. XX.27. Ошибка бустинга на отложенной выборке при увеличении числа базовых алгоритмов.

Интересное наблюдение: ранний останов есть в самых лучших современных алгоритмах: градиентном бустинге и нейронных сетях.

**9. Выбор модели / архитектуры алгоритма.** Мы уже поняли, что большинство приёмов упрощают используемую модель алгоритмов. Выбор модели – это также средство упрощения (вы можете не выбирать сложную). Как правило, модель подбирается под задачу. Например, если есть гипотеза о линейной зависимости целевого значения от значений признаков, то используются линейные методы, см. рис. XX.1 (слева) – здесь использование линейной модели уменьшит переобучение. Особенно это важно в глубоком обучении, где потенциально много архитектур нейронных сетей, но задачу пытаются решать только архитектурами, проверенными на аналогичных задачах. Многие из них содержат архитектурные особенности «под задачу»: свёртки для работы с изображениями, рекуррентные элементы или self-attention для работы с последовательностями и т.п.

## Вопросы и задачи

1. Мы проиллюстрировали с помощью экспериментов, что смещение может возрасти при увеличении сложности. Но может это связано с погрешностью эксперимента. Можно ли привести простой пример, который это иллюстрирует?
2. Для решающего дерева логично предположить, что увеличение глубины приводит к увеличению сложности. Могут ли в какой-нибудь задаче графики ошибок вести себя как на рис. XX.28, т.е. переобучение не увеличивается при увеличении глубины?

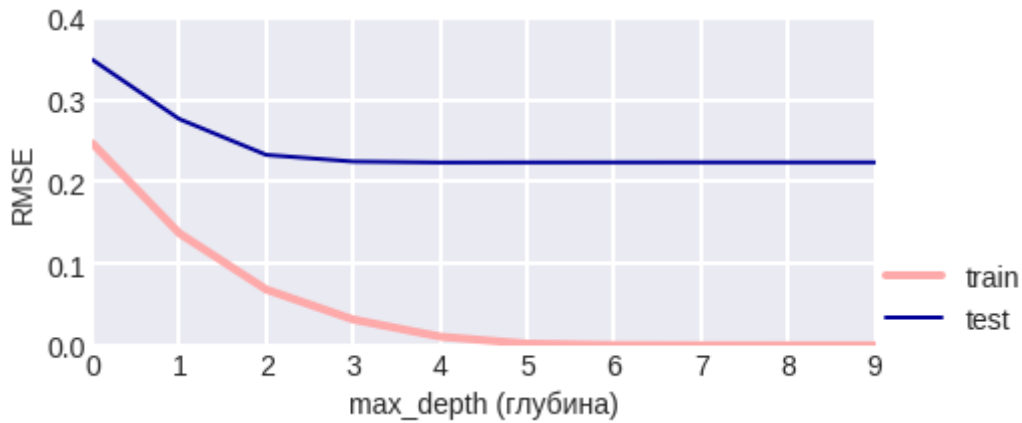


Рис. XX.28. Средние ошибки на обучении и тесте для деревьев разной глубины.

3. Предложите ещё способы аугментации для табличных данных.

### Итоги

- Матожидание ошибки =  $\text{смещение}^2 + \text{разброс} + \text{шум}$ .
- Увеличение сложности модели, как правило, приводит к увеличению разброса и уменьшению смещения.
- Выбор гиперпараметров часто нужен для определения оптимальной сложности модели.
- Для борьбы с переобучением работают с данными: выбирают, увеличивают объём, аугментируют, улучшают качество, уменьшают избыточность, используют дополнительные.
- Также для борьбы с переобучением влияют на модель: используют регуляризацию, валидацию, выбирают модель.

Спасибо за внимание к книге!  
Замечания по содержанию, замеченные ошибки  
и неточности можно написать в телеграм-чате  
<https://t.me/Dyakovsbook>